# SANDIA REPORT

SAND90—0589 • UC—410
Unlimited Release
Printed July 1990

# A Development Plan for a Massively Parallel Version of the Hydrocode CTH

A. C. Robinson, E. Fang, D. Holdridge, J. M. McGlaun



**Prepared by**
**Sandia National Laboratories**
**Albuquerque, New Mexico 87185 and Livermore, California 94550**
**for the United States Department of Energy**
**under Contract DE-AC04-76DP00789**

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.



Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

SAND90-0589
Unlimited Release
Printed July 1990

Distribution
Category UC-410

# A Development Plan for a Massively Parallel Version of the Hydrocode CTH

A. C. Robinson, E. Fang, D. Holdridge and J. M. McGlaun
Sandia National Laboratories
Albuquerque, New Mexico 87185

## Abstract

Massively parallel computers and computer networks are beginning to appear as an integral part of the scientific computing workplace. This report documents the goals and the corresponding development plan of the massively parallel project of Departments 1530 and 1420. The main goal of the project is to provide a clear understanding of the issues and difficulties involved in bringing the current production hydrocode CTH to the state of being portable to a number of currently available parallel computing architectures. In the process of this research, various working versions of the code will be produced.

# 1.0 Introduction

The transient dynamic continuum mechanics code, CTH, a "hydrocode", is used to analyze a wide variety of problems in shock wave physics. It is a very general purpose code which allows the use of many materials and complex configurations and is, in general, expensive to run. Budget and time pressures require that hydrocode computations be completed faster and more cheaply than in the past. At the same time, demands for capability to examine problems currently out of reach are ever present. Parallel processing architectures of various types give the promise of providing much more computing power at reduced cost.

A number of different parallel processing architectures are available. The classical shared memory supercomputers, such as the CRAY machines, have a small number of very fast CPU's (4-8) with vector pipe hardware which allows for very fast processing of arithmetic operations on arrays. The CPU's can either operate independently on separate jobs or work concurrently on the same job using multitasking algorithms which usually take the form of spreading work across outer DO loops. When a CPU is done with one loop, it returns to select the next loop which has yet to be processed. The CTH hydrocode currently runs either in single CPU or multitasking mode on a CRAY XMP 4/16 with extensive use of the Solid State Disk (SSD) for 3D calculations.

The MIMD (Multiple Instruction Multiple Data) machine such as the hypercube architectures offered by the NCUBE Corporation consists of many (up to tens of thousands) of medium speed CPU's. Each node runs asynchronously but work can be correlated by explicit message passing. Each node has its own local memory with enough memory for both the code and distributed data. Strictly speaking, the CRAY shared memory multitasking architecture falls under the general heading of a MIMD machine. However, for the purposes of this plan, MIMD refers to a distributed memory MIMD architecture.

The SIMD (Single Instruction Multiple Data) computer such as the Connection Machine offered by Thinking Machines Corporation takes the approach of defining vector or matrix data elements which are spread out among thousands of local memory computational elements or nodes which are arranged in a hypercube architecture. One to several data elements of each vector or matrix are stored on a node. A host computer executes "vector" instructions which are broadcast and executed in parallel on large data sets.

In addition to the massively parallel machines mentioned above, one has the additional option of computing on networks. For example, a small SUN LAN may include 8 different workstations each of which represent significant computational capability. Together they represent a very powerful compute engine and one would wish to utilize this capability. A number of software and hardware products are under development with the goal of turning LANs into powerful distributed memory MIMD computing environments. One example is LINDA, developed at Yale University.

Each of the above computing concepts is favored by their respective proponents based on arguments of cost, raw speed and usability. The best architecture is certainly not an absolute measured in terms of raw speed but depends on the needs and sophistication of the

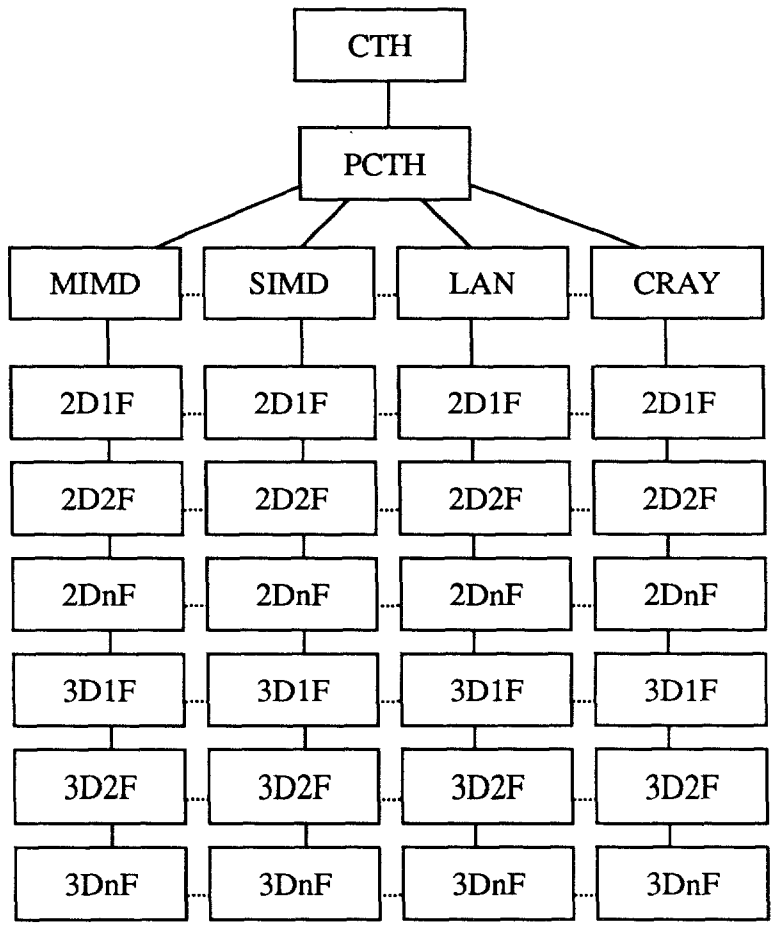
user, the size and extent of natural parallelism of his problem and, most importantly, the user's budget in time, frustration and money. The user's budget is a primary concern and is addressed mainly in terms of the quality of software and attendant customer support. A truly viable "production" machine provides an environment with costs which are low enough and/or the benefits high enough to be worth the effort to a sufficient number of users to support the system.

This report describes a set of goals and an evolutionary development method which is intended to port the code CTH to the various architectures mentioned above and to provide feedback on the structural changes which will be required in the code to provide portability across the various compute machines. This is especially important as massively parallel computing architectures are evolving rapidly. No standard hardware designs or standard software interfaces have emerged. Some degree of portability is essential to preserving the value of the parallelization project. We allow that one possible conclusion of the research may be that the goal of portability or near-portability across the broadest class of architectures is not feasible or desirable. The intent of code portability is to conserve investment in the code itself while at the same time providing added-value by permitting the code to be run economically and reasonably efficiently on each of the above compute engine prototypes. It is recognized that major database and coding changes may be necessary to achieve a near portability goal. Load balancing difficulties for the CTH code are expected to be severe at first. However, load balancing strategies will be implemented in an incremental way as the scope and type of problems are identified and as familiarity with the new computing paradigms increases.

## 2.0 Parallel CTH Phased Development Plan

It is proposed that the development of the massively parallel version of CTH follow an incremental or evolutionary development program. The developing code will be termed PCTH for Parallel CTH. The evolutionary development is intended to minimize programming clutter while leaving open the essential code structure issues. The current SUN version of CTH will be modified to remove physics options which have little intrinsic bearing on the database, coding and load balancing issues which are the essence of the research and development this project is expected to address. A progressive set of codes will be implemented. The evolutionary development process with its corresponding acronyms is shown in Figure 1.

**Figure 1. Evolutionary PCTH development sequence.**




The code development for the two major parallel architectures under consideration, MIMD and SIMD, will proceed somewhat concurrently in order to make comparisons on an equal basis as the complexity of the coding and algorithms increases. Every attempt will be made to keep the coding style and database structures the same. A particular code version and type will be denoted by {name}.{machine}.{phase}.{version}. The {name} entry is PCTH. The entry {machine} is one of MIMD, SIMD, LAN or CRAY. The entry

{phase} is one of 2D1F, 2D2F, 2DnF, 3D1F, 3D2F or 3DnF where 2D stands for a two dimensional code and 3D stands for a three-dimensional code. 1F denotes a single fluid code, 2F denotes a two fluid version of the code and nF denotes a multiple fluid version of the code. The entry, {version}, is a whole number which may be followed by a descriptive comment. The zero value refers to the first rewrite of the standard production code which runs correctly on the target architecture (either a simulator or a real machine) at the specified phase. Higher version numbers will be assigned to subsequent major upgrades to the code at a given phase.

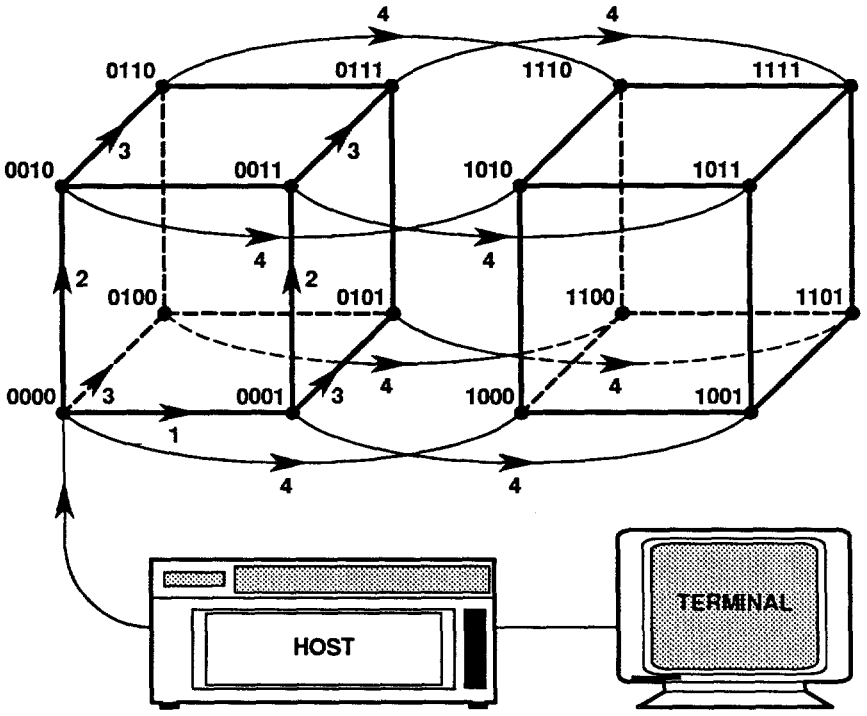
A number of options currently available in the production version of CTH will not be available in the initial evolving versions of PCTH. These options will be removed initially because they would provide only additional programming difficulties and no new conceptual architectural and algorithmic barriers. A few of the options which will not be present in the initial version of PCTH are: 1) deviatoric stresses (only compressible fluids will be considered), 2) high explosives, 3) user defined energy sources and 4) tracer particles.

This development plan explicitly allows for skipping or terminating the development of various code configurations if it becomes clear that a continued detailed investigation is unnecessary or unwise. At first glance the development sequence described may appear to represent a totally unreasonable amount of work. Fortunately, the MIMD message passing computing paradigm maps in a straightforward way to the LAN architecture so that in this case we have only to obtain or develop message passing software equivalent to the message passing libraries found on integrated distributed memory MIMD machines. The reason the LAN architecture is not included under the general MIMD heading is that the communication time issues and solutions may of necessity be different than those for a MIMD machine with very rapid communications. One could also utilize the CRAY in a MIMD message passing mode since multiple CPU's could be employed for different node processes. However, it is possible that the MIMD paradigm would be more effectively utilized for purposes of reducing memory charges via extensive use of the SSD. The major development effort for this project will revolve around the MIMD and SIMD architectures.

# 3.0 The MIMD Architecture

Our prototype MIMD architecture is the hypercube. The hypercube is an inductively defined architecture which has $P = 2^d$ node processors and provides user access through an additional host or front end machine. The set of node computers is called an order $d$ hypercube. The configuration is illustrated in Figure 2. The hypercube has the property of requiring that at most $d$ communication paths be traversed in order to send a message from one node to any other node.

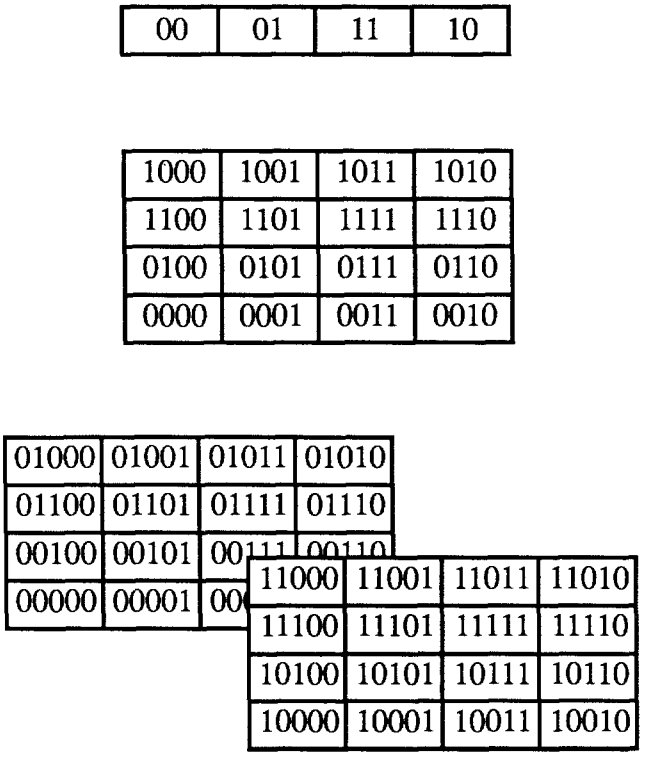
**Figure 2. Typical hypercube configuration.**




The host machine can operate as an I/O server for the hypercube and can be any standard serial computer. It provides terminal access, user disk space, and utility software to access the hypercube. Each node has a limited local operating system which is capable of loading and running executables passed from the host and of passing messages (data) between itself and the host and/or any of the other nodes. Each node operates asynchronously but can be synchronized by explicit message passing techniques such as blocked reads. The nodes may have high speed graphics output as well as local direct disk storage.

Physical domains represented by a Cartesian grid can be mapped to a hypercube in a straightforward way using binary reflected Gray code. This mapping results in physical neighbors becoming "next door"computer neighbors and is given in Figure 3. In two dimensions physical neighbors are at most two processors away in computer space. Similar-

**Figure 3. One, two and three dimensional Gray codes**




ly physical neighbors in three dimensions are at most three processors away. This standard Gray code decomposition is clearly applicable to CTH which has a rectangular parallelipiped mesh scheme.

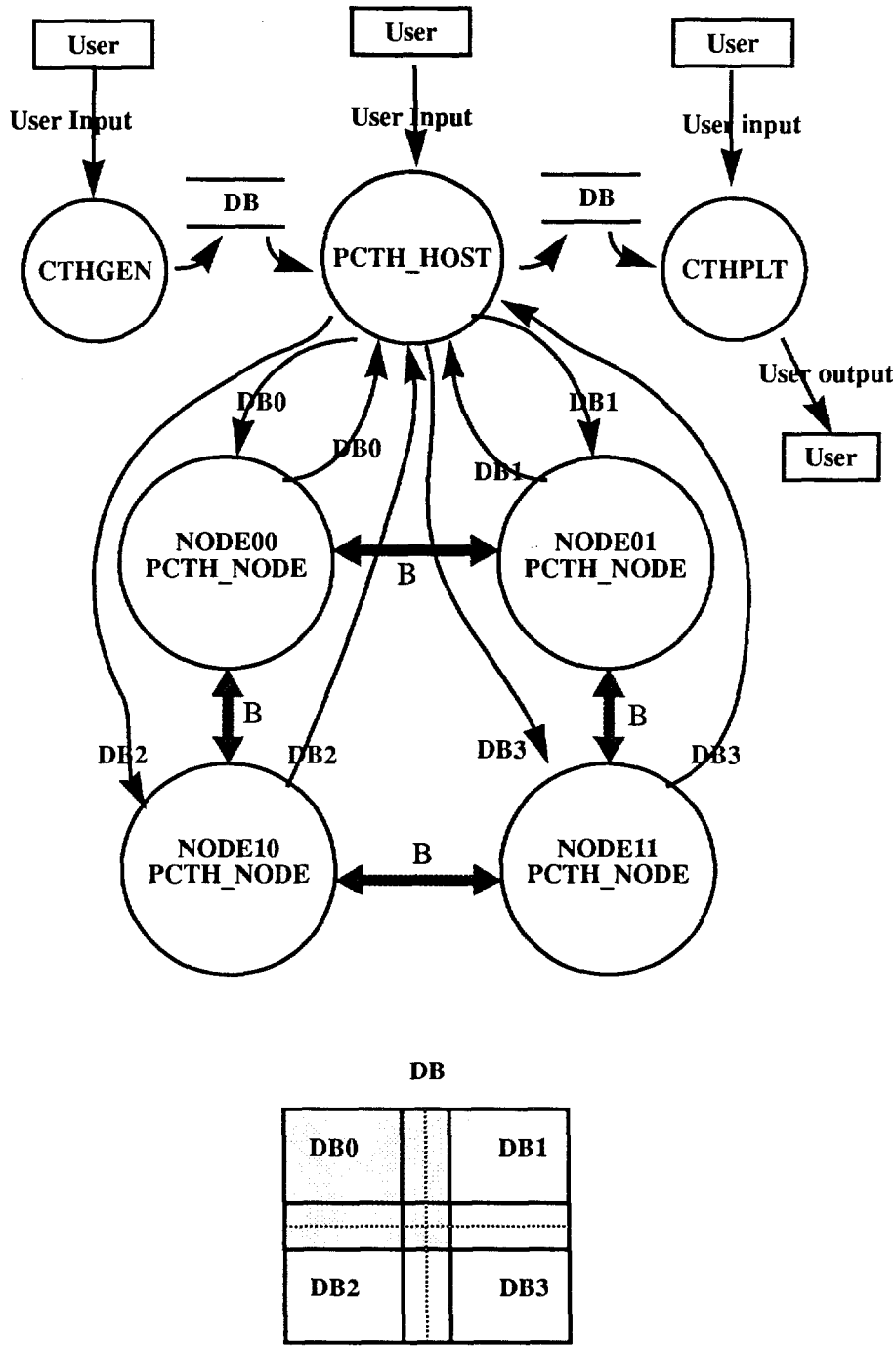
This decomposition also has application in the case of the SIMD machine. However, in the MIMD case each processor refers to a block of data and in the SIMD case we think of each processor as containing only one computational cell.

## 3.1 PCTH.MIMD.{phase}.0 Conversion

The code PCTH.MIMD.{phase}.0 is a straightforward port of CTH to the hypercube architecture using a Gray code mapping (described in the previous section) to split up the data and the work. This approach by itself will not necessarily suffice for obtaining good load balancing, but represents a way of rapidly obtaining useful information and comparisons between different architectures and of providing a basis upon which to implement various load balancing strategies. The changes that will be required to create the version 0 codes are described below. The version 0 dataflow diagram is shown in Figure 4 . The symbol DB stands for "database." The figure also shows how boundary quantities are shared between the nodes. The boundary-passing libraries will be built to be as general as possible so as to be reusable for future development. For example, variable numbers of boundary cells may be required to accommodate advanced numerical schemes, such as the essentially non-oscillatory (ENO) fluxing methods. Paired reads and writes of boundary information to neighboring nodes may be accomplished, for example, using a red/black or

checkerboard scheme to avoid communication bottlenecks and to ensure rapid and robust communications. The code will be structured so as to allow variations in the type (e.g. strips versus blocks) of subgrid decomposition selected.

**Figure 4. Version 0 sample dataflow diagram and boundary overlap for 4 nodes.**



## 3.2 Code Changes to Create PCTH.MIMD.{phase}.0

The next several pages give a general outline of the current CTH top level executive routine and the replacements or substitutions (x=>y means module x is replaced by module or action y) which will be necessary in order to create PCTH.MIMD. The notation (=>x) signifies that a totally new module named x must be built.

CTH =>PCTH

cth=>pcth

fndid = tell db manager this is cth
cedrv = perform calculation

CEDRV - TOP EXECUTIVE ROUTINE

Cedrv then calls several routines. Most of them are small routines. The routine that performs most of the work is celop=>celoph.

cedrv

cemuli = start multitasking library
syctty = open terminal i/o files
sysecd = zero timers
cever = read all package versions
fndinr = first pass through user input
logmes = messages to log file
sydate = get date
sytime = get time
syjobn = get job name
logmes = messages to log file
cever = write all package versions
dbmzer = zero storage flags
dcofbk = multitasking memory manager
dbaget = multitasking memory manager
dbmmxk = set maximum number of in-core planes
scclv = set number of extra variables per cell
uinz = read user input that require no database in core
dbgdbs = read initial short database
uins = read user input that require small database
fatret = check for bad input
dbgdbl = read rest of database
dbmclp = clear any planes left in core
cepbd = initialize data
scdrv = define scratch storage requirements
uinf = read user input that requires entire database
fatret = check for bad input
logmes = messages to log file

(celop = main program loop) => (celoph = host driver)
dcptal = write all planes to ssd
dbpdb = write restart file
setprio= create dropfile on ctss system
dbmclp = clear planes out of memory
logmes = messages to log file
syftim = write timing information to log file
sdclos = close and destroy ssd file
massit = copy files to archival storage
logmqe = copy executive log file to output file
fishit = make microfiche if requested

CELOP=> CELOPH

Celoph is the executive routine that sets up the hypercube, subdivides the database and collects information as it comes back from the nodes. For some architectures celoph does nothing more than call CELOPN. A possible pseudo-code version of celoph is given below.

```
celoph
    if (first cycle of problem) then
        cezero = perform cycle zero tasks
        dtcalc = calculate initial timestep
    endif

    =>ceinit = attach and load hypercube
    =>cesrom= broadcast common databases to nodes.
    =>cesplt = split up physical database and send to nodes one at a time.
    =>cepoll = read messages and act on them from the host while not done
        read error messages
        read and tally all_done messages
        read and process efficiency and timing information
        read database from nodes and write to permanent storage
        if(something_is_very_wrong) do error processing
        if(all_done && no_more_messages) exit
    end cepoll
    =>ceclos=detach hypercube
end celoph
```

CELOP => CELOPN

Celopn is the executive subroutine that calls all the major packages. It is the only routine called by the main node program NCTH. There is one loop through the major packages. The loop continues until the routine ce2stp passes back a nonzero flag. The nonzero flag indicates the calculation should stop. The pseudo-code version of celopn is given below.

```
cemul = initialize multitasking=>node initialization and database read
    begin cemul
    =>cerrom=read and broadcast common blocks
    =>nodmem=allocate memory and set up pointers
    =>cerecv=get database for this node
    =>neighb=compute neighbor addresses
    =>bdinit=fix up some arrays for use by boundary cells
    end cemul
edstar = generate initial edit=>delete
start of main loop
    ceint = check for user interrupts
    ce2stp = check for end of calculation
    if ( end of calculation ) then
        edende = generate final edit=>cesend=send restart
        cemule = shutdown multitasking=>"node done" message to host
        return calling routine=>return to ncth and stop
    endif
    eddrv = generate edit if desired=>send restart to host if desired
    =>cebdup=update boundary cells
    celag = Lagrangian step
    erdrv = Eulerian remap step
    radman = Radiation step=>delete
    vddrv = rigid body velocities=>delete
    edln = one line edit per cycle=>delete
    dtcalc = new timestep, minimize timestep over all nodes
    sysecd = get total cpu time=>get, compute and send to host timing info
end of main loop
```

## 3.3 PCTH.MIMD.{phase}.{version}

Version numbers greater than zero for each phase represent additional milestones in terms of increasing the options, robustness and capabilities of the version 0 product. For example, the first version 0 MIMD code will simulate only problems which are fully periodic in both the x and y directions. The periodic case is the natural boundary condition for a Gray code domain decomposition. Subsequent versions of the code will examine issues of communications and boundary update schemes. The physics routines will be modified to take the non-periodic boundary condition options out of the field solver routines and into boundary update routines. The first versions of the code for some phases are not expected to result in extremely good efficiencies. A block-structured grid approach may permit effective static and dynamic load balancing algorithms to be developed. Another

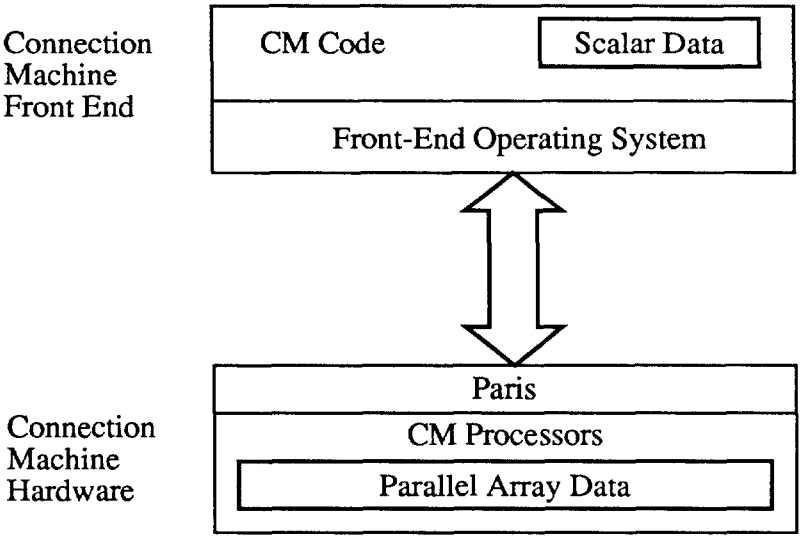
load-balancing strategy which might be implemented for some problems and code phases is a scattered decomposition on the equation-of-state iterations or mixed cell computations. One such algorithm in this regard has already been given in terms of a task list scatter[6]. New CTH physics algorithms as well as specific dynamic load balancing algorithms will be considered.

Currently, there are no known hydrocode performance ratings on hypercubes. Initial efficiency ratings will vary wildly depending on the problem being solved and the number of nodes in use. For example, the code PCTH.MIMD.2D1F.0 may run fairly efficiently for some problems, while the code PCTH.MIMD.3DnF.0 is certain to have serious load imbalances due to the extra work required at multiple fluid interfaces. Efficiency ratings varying from 1 to 90 percent would not be surprising. Load balancing strategies which resulted in at least an 80 percent efficiency rating over a wide variety of reasonable production level problems would be considered an outstanding result.

# 4.0 The SIMD Architecture

The SIMD massively parallel computing paradigm differs dramatically from the MIMD hypercube model. We take as our target architecture the Connection Machine (CM-2) produced by Thinking Machines Corporation. The CM-2 combines up to 65,536 ($2^{16}$) 1-bit processors in a hypercube architecture. Each processor has 8 Kbytes of memory associated with it. However, in the SIMD case the executable code does not reside in the processor memory but only the data resides there. Instructions act in parallel on data elements associated with each processor. The user's code actually executes on the front end machine and instructions to act on the parallel data are broadcast from the front end machine. Floating point computations are performed by Weitek floating point chips. There are 32 processors assigned to each Weitek chip. An important aspect of the Connection Machine is the concept of virtual processors. Transparently to the user, more data elements of a given type can be assigned than the actual number of physical processors which are present on the machine. Thus two or more data elements from a single parallel data structure are assigned to a given processor. The Connection Machine configuration is illustrated in Figure 5.

**Figure 5. Code and Data Distribution on the Connection Machine.**

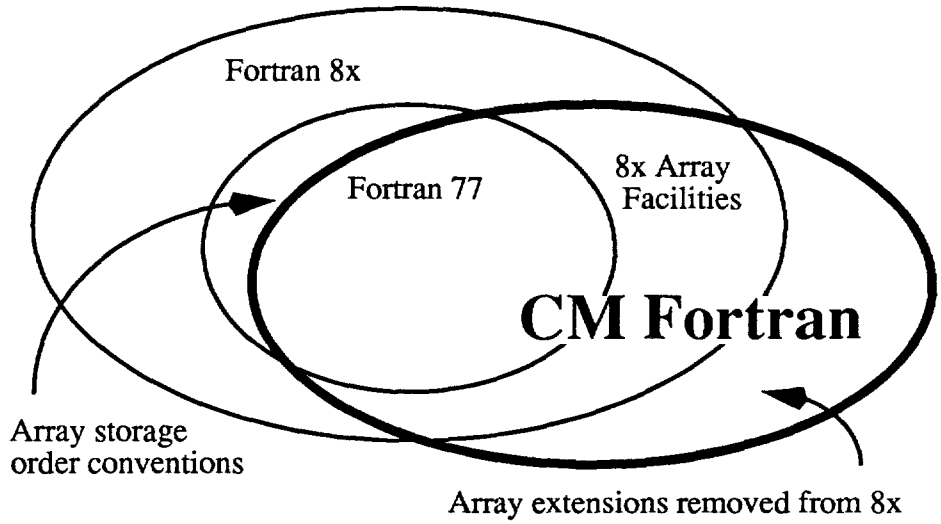



## 4.1 PCTH.SIMD.{phase}.0 Conversion

The CTH code will have to be modified extensively to run on the Connection Machine. These modifications consist principally of rewriting subroutines into vector operation syntax and of modifying the memory management and subroutine call structures to accommodate the data-parallel computing paradigm. It is expected that these structures will also be acceptable for the MIMD hypercube architecture, so that it will be possible to develop a code philosophy which will be essentially the same on both the MIMD hypercube and the SIMD machine albeit with greatly differing syntax. Development of a version 0 Connection Machine code will require a great deal of recoding in the CTH physics routines. This is not the case with the MIMD architecture.

R. Smith and J. Baumgardner at Los Alamos have demonstrated that single material and multiple material hydrocode algorithms can be ported effectively to the Connection Machine[1]. It is quite clear from their results that the Connection Machine has the potential to be a major contender in the supercomputing arena. They chose to implement their application using CM Fortran (a Fortran 8x type compiler). The conversion process appears to be fairly straightforward and has the added benefit of code conciseness and clarity resulting from the Fortran 8x array syntax. Backwards compatibility with standard Fortran 77 will not be possible. A preprocessor of some sort would be required to rewrite the vector constructs into Fortran 77. We know of only one such product in commercial development and none on the market. Effectively, this means that for our research purposes we must live with a MIMD and a SIMD version of the code until such time as pre-processors or compilers are available which could deal with the syntax compatibility issue. We are aware of at least three Fortran compilers which have array syntax capability. These are the Microsoft 5.0 PC compiler, the CRAY CFT77 compiler and the CM Fortran compiler. We will consider the CM Fortran to be the standard for the purposes of our research. The Venn diagram outlined in Figure 6 shows the relationship between the Fortran 8x standard as currently specified, Fortran 77 and CM Fortran.[3].

**Figure 6. Relationship of CM Fortran to other Fortrans.**




An important feature of the above diagram is that the standard Fortran 77 storage order conventions for arrays, which are often used by programmers for various reasons of efficiency and convenience, are no longer valid for arrays stored on the Connection Machine.

Currently, the Sandia Connection Machine has 16K nodes with 8K bytes of memory each. This machine is acceptable as a research platform but has a major limitation in that it can only do 32 bit single-precision arithmetic on its Weitek floating point chips. Weitek 64 bit chips are available and could be obtained with sufficient funds. CTH must have double precision arithmetic to do production level calculations. However the lack of 64 bit floating point chips is not seen as an impediment for this project. Limited sample double-precision computations can be performed at Thinking Machines Corporation over the network. The development of both the MIMD and SIMD codes will be done here at San-

dia during concurrent time frames so as to be able to make reasonable comparisions between the two architectures.

For our development work we consider reliable and effective access to the 1400 open network from the secure area to be a critical need item. The current VAX 750 MILNET interface is overloaded and effectively inoperable as a means of interactively accessing other machines on the Internet. The UNIX 8530 upgrade to the current system is scheduled for August 1990 and should alleviate the current situation as local access will then be available to the research machines maintained by Org. 1400.

# 5.0 LAN and CRAY Virtual Hypercubes

The development of a hypercube version of CTH should also lead fairly easily to additional applications on more conventional compute platforms. These platforms are not considered to be the primary target of this research and development project but an effort will be made to provide portability to these additional platforms.

The MIMD hypercube code will clearly be applicable to some extent to a network architecture. The network we are thinking of in particular is a group of 2, 4 or 8 workstations on a local area network which are able to be dedicated for an overnight run on a single computation. It is obvious that the communications to computation ratio is going to be much larger in the network case than for a dedicated hypercube. For this reason, it seems reasonable to envision a small number of workstations running applications for which the communication to work ratio can be made small enough and the load balancing difficulties minimized to the point of providing reasonable speedups. The workstation hypercube network also provides an interesting environment for hypercube code algorithm development as it will tend to greatly penalize communication between the nodes. The idea of using a network as a virtual hypercube is not a new idea. Portable environments have been used at Argonne National Laboratory for some time[2][5]. Several such libraries are in existence and Argonne is currently developing a Sun network send/receive library based on the UNIX remote procedure calls (RPC) technology[4]. A network capability may be very important in the sense of being able to better serve customers who cannot pay for compute time on a more powerful machine.

The hypercube code architecture may also be considered as a prototype for a decomposition scheme for running very large problems on a CRAY machine with solid state disk access. The standard CTH database would be divided up into hypercube blocks and then these individual blocks would be run sequentially (and possibly concurrently on separate processors). The data would be swapped out through the solid state disk. In this way one would be able to not only reduce memory charges but also to run computations which would not have been possible otherwise. A possible way to pass information between virtual nodes is through solid state disk files. The precise means of passing this information would be hidden in the boundary subroutines.

# 6.0 Summary

We have described a development plan for constructing a massively parallel version of the hydrocode CTH. This project is intended to demonstrate techniques for porting the code to a variety of compute platforms including MIMD hypercubes, the SIMD Connection Machine and secondarily to workstation local area networks and CRAY machines. The parallel extensions and/or modifications to the code which prove successful are intended to become part of the future CTH production environment. The development plans described herein are based on our current knowledge of parallel hardware and software capabilities and will clearly be modified as the project progresses and additional options become available.

# 7.0 References

[1] Baumgardner, J. and R. Smith, Personal Communication, Los Alamos National Laboratories, January 1990.

[2] Boyle, J., et. al., Portable Programs for Parallel Processors, Holt, Rinehart and Winston, Inc., New York, 1987.

[3]CM Fortran Reference Manual, Ver. 5.2-0.6, Thinking Machines Corporation, Cambridge, Massachusetts, 1989.

[4] Leibfritz, D., Argonne National Laboratories, Personal Communication, 1990.

[5] May, E. N., Portable Parallel Programming in a Fortran Environment, Argonne National Labs, ANL-HEP-CP--89-50, 1989.

[6] Robinson, A. C., A Scattered Decomposition Algorithm for Load Balancing Task List Calls on a MIMD Hypercube, In preparation, 1990.

**Distribution:**

Sandia internal:

1400 E. H. Barsis
1410 P. J. Eicher
1412 P. A. Erickson
1412 G. S. Davidson
1412 C. F. Diegart
1420 W. J. Camp
1421 R. J. Thompson
1421 D. B. Holdridge(5)
1422 R. C. Allen
1424 R. E. Benner
1424 J. N. Jortner
1424 S. J. Plimpton
1424 M. P. Sears
1424 J. P. VanDyke
1424 C. T. Vaughan
1500 E. H. Barsis (Acting)
1510 J. W. Nunziato
1520 L. W. Davison
1530 J. R. Asay
1530 S. L. Thompson
1531 M. Elrick
1531 L. N. Kmetyk
1531 J. M. McGlaun(5)
1531 K. Budge
1531 M. Elrick
1531 E. S. Hertel
1531 L. N. Kmetyk
1531 J. S. Peery
1531 A. C. Robinson (10)

1533 P. Yarrington

1533 R. L. Bell

1533 W. T. Brown

1533 P. J. Chen

1533 E. Fang(5)

1533 A. Farnsworth

1533 G. I. Kerley

1533 M. E. Kipp

1533 S. T. Montgomery

1533 F. R. Norwood

1533 J. S. Rottler

1533 S. A. Silling

1534 J. A. Ang

1534 L. C. Chhabildas

1534 J. E. Dunn

1534 M. D. Furnish

1534 D. E. Grady

1534 M. Shahinpoor

1534 J. W. Swegle

1534 T. G. Trucano

1534 J. L. Wise

1550 C. W. Peterson

1600 W. Herrmann

3141 S. A. Landenberger (5)

3151 W. L. Garner (3)

3154-4 C. H. Dalin, for DOE,TIC (28)

| Org. | Bldg. | Name | Rec'd by | Org. | Bldg. | Name | Rec'd by |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |


Sandia National Laboratories